Amendments to the Claims:

This listing of claims will replace all prior versions, and listings, of claims in the application:

Listing of Claims:

1-23. (Cancelled)

24.   (Currently Amended) A process for assigning tasks to a processor in a multiprocessor digital data processing system having a preemptive type operating system and a given number of processors capable of processing said tasks in parallel, ~~comprising,~~ comprising:

in at least one preliminary phase, dividing said given number of processors into a plurality of processor groups, each ~~group~~ of the plurality of processor groups comprising ~~predetermined numbers~~ a number of ~~processors,~~ processors:

creating a plurality of elementary task queues:

associating each of the plurality of elementary task queues with a different one of the plurality of processor groups:

~~dividing said tasks into a predetermined number of elementary task queues and storing a predetermined number of tasks to be processed in a given order of priority in each elementary task queue, each of said processor groups being associated with an elementary task queue, each of the stored predetermined number of tasks being associated with one of the processors associated with said elementary task queue.~~

storing a plurality of tasks to be processed in one of the plurality of elementary task queues, wherein the plurality of tasks are stored in a given order of priority; and

assigning each of the plurality of tasks to one of the given number of processors in the associated processor group.

25.   (Currently Amended) A process according to claim 24, wherein said plurality of processor groups each comprise an identical number of processors.

26.   (Currently Amended) A process according to claim 24, additionally comprising generating a series of tests and measurements in an additional preliminary phase for determining the number of processors in each processor group and the number of groups for achieving the best performance of said system.

27.   (Currently Amended) A process according to claim 24, wherein ~~the architecture of~~ said system is of the non-uniform memory access type (NUMA), and the system comprises a predetermined number of modules linked to one another, wherein each module ~~comprising~~ comprises ~~a given number of processors~~ one of the plurality of processor groups and storage means, ~~each of said modules constituting one of said processor groups,~~ and wherein each module ~~being~~ is associated with one of said elementary task queues ~~of an associated processor~~.

28.   (Currently Amended) A process according to claim 24, further comprising associating each of said processors with a first data structure for identification of the associated processor, said first data structure comprises at least one first set of pointers, associating said first set of pointers with one of said elementary task queues, associating each of said elementary task queues with a second data structure, said second data structure having at least one second set of pointers, associating said second data structure with one of said plurality of processor groups, storing all of the tasks to be processed in said system in a table, each of said second data structures of the elementary task queues further comprising a third set of pointers, said third set of pointers each associating

elementary task queues with one of said tasks stored in the table or with a series of concatenated tasks, and associating each of said tasks of the table with a third data structure that comprises a fourth set of pointers said fourth set of pointers associating said third data structure with one of said elementary task queues.

29.   (Currently Amended) A process according to claim 24, further comprising distributing said tasks among said plurality of elementary task queues in at least one additional phase by searching, when a new task to be processed is created, for a queue with the lightest load among all of said elementary task queues of said system and assigning said new task to said elementary task queue with the lightest load so as to balance the a global load of said system among said elementary task queues.

30.   (Previously Presented) A process according to claim 29, further comprising performing said distribution of tasks by determining a composite load parameter associated with each of said elementary task queues associating each processor with a memory, calculating said composite load parameter as the sum of the load of a processor or a processor group associated with said elementary task queue and the load of the memory associated with said processor or processor group.

31.   (Currently Amended) A process according to claim 29, further comprising checking in a preliminary step whether said new task is linked to one of said elementary task queues, and when said test is positive, assigning said linked task to the elementary task queue.

32.   (Currently Amended) A process according to claim 24, further comprising at least one additional phase and of searching for a remote elementary task queue that is not empty when one of said elementary task queues associated with one of said processor groups is empty of executable tasks, selecting in said empty task elementary queue a task executable by one of said processors of said processor group associated with the empty elementary task queue and transmitting said selected task to said one of said processor for processing so as to globally balance the processing of said tasks in said system.

33.   (Previously Presented) A process according to claim 32, wherein said non-empty elementary task queue has a predetermined minimal occupation threshold.

34.   (Previously Presented) A process according to claim 33, further comprising storing the tasks in decreasing order of priority, skipping a predetermined number of tasks before scanning the other tasks of said non-empty elementary task queue in order to search for an executable task and have said executable task processed by one of said processors of said processor group associated with the empty elementary task queue.

35.   (Currently Amended) A process according to claim 34, wherein said number of skipped tasks and the maximum number of scanned tasks among all tasks stored in said non-empty elementary queue are variable over time and are determined by a self-adapting process from the number of tasks that are or are not found during said scans and from the position of these tasks, sequenced in order of priority, in said non-empty elementary queue.

36.   (Currently Amended) A process according to claim 32, wherein said selected task is associated with a minimal value of a cost parameter, which measures global performance degradation of said system due to the processing of said selected task in said non-empty remote elementary queue by one of said processors of said processor group associated with the empty elementary queue.

37.   (Previously Presented) A process according to claim 24, further comprising periodically measuring for a balanced distribution of said tasks in said elementary task queues in at least one additional phase and when an unbalanced state of said system is determined, selectively moving tasks from at least one task elementary queue with a heavier load to an elementary task queue with a lighter load.

38.   (Previously Presented) A process according to claim 37, further comprising discontinuing the step of selectively moving tasks when said imbalance is below a certain threshold.

39.   (Currently Amended) A process according to claim 37, wherein all or some of said tasks belong to multitask processes, and each multitask process requires a given memory size and workload, further comprising measuring workloads and memory sizes, in the system and selecting the process requiring the greatest workload and the smallest memory size, and moving all the tasks of said selected process to the elementary queue with the lightest load.

40.   (Currently Amended) A process according to claim 39, further comprising a preliminary step of checking whether all tasks of said multitask process that must be moved belong to the elementary task queue set with the heaviest load and whether any task is linked to any of said processor groups.

41.   (Previously Presented) A process according to claim 24, wherein said preemptive operating system is used in a server in a distributed network environment.

42.   (Currently Amended) Architecture for a multiprocessor digital data processing system, comprising a given number of processors, for implementing a process for assigning tasks to be processed to said processors, said system having a preemptive operating system and a given number of processors capable of processing said tasks in parallel, said processors being divided, in a preliminary phase, into groups of processors, and;

          wherein, an elementary task queue is subsequently created and associated with each of the groups, of processors; and

          each of said elementary task queues storing stores a predetermined number plurality of tasks to be processed in a given order of priority, so that each of the stored predetermined number plurality of tasks of each of said elementary task queues is associated with one of the processors in the group of processors associated with the elementary task queue.

43.   (Previously Presented) Architecture according to claim 42, further comprising means for determining the load of said elementary task queues and for assigning a new task created in said system to the elementary task queue with the lightest load.

44.   (Previously Presented) Architecture according to claim 42, further comprising, when one of said elementary task queues associated with one of said processors is empty, means for locating a non-empty, remote elementary task queue and an executable task in said non empty elementary task queue, and assigning said executable task to said one of said processor for processing said executable task.

45.   (Previously Presented) Architecture according to claim 42, further comprising means for detecting an imbalance between elementary task queues, and for determining when an imbalance is detected the elementary task queue with the heaviest load and the elementary task queue with the lightest load, and means for moving tasks from the elementary task queue with the heaviest load to the elementary task queue with the lightest load.

46.   (Previously Presented) Architecture according to claim 42, wherein the operating system of the processing system is of the nonuniform memory access type (NUMA), and comprises modules linked to one another, each module comprising a given number of processors and storage means, each of said modules constituting one of said groups, each module being associated with one of said elementary queues.

47.   (Previously Presented) Architecture according to claim 43, wherein the operating system of the processing system is of the nonuniform memory access type (NUMA), and comprises modules linked to one another, each module comprising a given number of processors and storage means, each of said modules constituting one of said groups, each module being associated with one of said elementary queues.

48.   (Previously Presented) Architecture according to claim 44, wherein the operating system of the processing system is of the nonuniform memory access type (NUMA), and comprises modules linked to one another, each module comprising a given number of processors and storage means, each of said modules constituting one of said groups, each module being associated with one of said elementary queues.

49.   (Previously Presented) Architecture according to claim 45, wherein the
operating system of the processing system is of the nonuniform memory access type
(NUMA), and comprises modules linked to one another, each module comprising a given
number of processors and storage means, each of said modules constituting one of said
groups, each module being associated with one of said elementary queues.